Министерство образования Республики Беларусь
Учреждение Образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра электронных вычислительных средств

Лабораторная работа № 6
«ИЗУЧЕНИЕ ГРАДИЕНТНЫХ МЕТОДОВ РЕШЕНИЯ ЗАДАЧИ НЕЛИНЕЙНОГО ПРОГРАММИРОВАНИЯ»

Выполнили:
ст. гр. 850702
Маковский Р. А.
Турко В. Д.

Проверил:
Станкевич А. В.

Минск 2020

ЦЕЛЬ РАБОТЫ:

Изучить градиентные методы решения задачи нелинейного программирования.

ИСХОДНЫЕ ДАННЫЕ(ВАРИАНТ 3):

ЭВС состоит из 2 блоков: процессорного и цифро-аналогового. Вероятности отказов блоков за заданную наработку Q_n=5 10^{-2} и $Q_{ца}$=2 10^{-2}. Потребляемая мощность P_n=3 Вт, $P_{ца}$=5 Вт. Требуется найти оптимальный резерв для каждого блока k_j для заданной общей вероятности отказов ЭВС $Q_0 \le 2\ 10^{-3}$ и минимальной потребляемой мощности. Резерв должен быть нагруженным.

$$Q_0 = \sum_{j=1}^{m} Q_j^{k_j+1}$$, k_j - кратность резервирования; m – число подсистем.

Init data

```
q0 = 0.002;
q1 = 0.05;
q2 = 0.02;

QRestriction := q1^(x1+1) + q2^(x2+1);

p1 = 3;
p2 = 5;

TargetF := p1 * (x1 + 1) + p2 * (x2 + 1);
```

ХОД РАБОТЫ:

Used formulas

```
EvaluateRestriction[x1_, x2_] := q1^(x1+1) + q2^(x2+1);

GradQ := Grad[QRestriction, {x1, x2}];
         градиент функции
EvaluateGradQ1[value1_] := Assuming[x1 == value1, Refine[GradQ[[1]]]];
                           предполагая             уточнить
EvaluateGradQ2[value2_] := Assuming[x2 == value2, Refine[GradQ[[2]]]];
                           предполагая             уточнить
AbsGradQ[x1_, x2_] := Sqrt[EvaluateGradQ1[x1]^2 + EvaluateGradQ2[x2]^2];
                      квадратный корень

EvaluateTargetF[x1_, x2_] := p1 * (x1 + 1) + p2 * (x2 + 1);

GradTargetF := Grad[TargetF, {x1, x2}];
               градиент функции
EvaluateGradTarget1[value1_] := Assuming[x1 == value1, Refine[GradTargetF[[1]]]];
                                предполагая             уточнить
EvaluateGradTarget2[value2_] := Assuming[x2 == value2, Refine[GradTargetF[[2]]]];
                                предполагая             уточнить
AbsGradF[x1_, x2_] := Sqrt[EvaluateGradTarget1[x1]^2 + EvaluateGradTarget2[x2]^2];
                      квадратный корень
```

Output functions

```
PlotAllowedPoints[x1_, x2_] := Module[{i = 0, j = 0, array = {{0, 0}}},
  For[i = 0, i ≤ x1, i++,
   For[j = 0, j ≤ x2, j++,
     AppendTo[array, {i, j}];
    ];
  ];
  ListPlot[array, PlotStyle → Black]
 ];

PlotRestrictionArea[max1_, max2_] := RegionPlot[QRestriction ≤ q0, {x1, 0, max1}, {x2, 0, max2}, PlotLegends → {QRestriction}, AxesLabel → Automatic,
  PlotStyle → LightGray, BoundaryStyle → LightGray];

PlotAllowedArea[max1_, max2_] := Show[PlotRestrictionArea[max1, max2], PlotAllowedPoints[max1, max2]];

PointsLength[x1_, y1_, x2_, y2_] := Sqrt[(x2 - x1)^2 + (y2 - y1)^2];

PrintResult[k_, init1_, init2_, x1_, x2_] := Module[{ceil1 = Ceiling[x1], ceil2 = Ceiling[x2], floor1 = Floor[x1], floor2 = Floor[x2], ans1 = -1, ans2 = -1},
  Module[{p1, p2, p3, p4, l, ansL},
   p1 = {ceil1, ceil2};
   p2 := {ceil1, floor2};
   p3 := {floor1, floor2};
   p4 = {floor1, ceil1};
   l := {PointsLength[x1, x2, p1[[1]], p1[[2]]], PointsLength[x1, x2, p2[[1]], p2[[2]]], PointsLength[x1, x2, p3[[1]], p3[[2]]], PointsLength[x1, x2, p4[[1]], p4[[2]]]};
   ansL = l[[1]];
   If[EvaluateRestriction[p1[[1]], p1[[2]]] ≤ q0,
    ans1 = p1[[1]];
    ans2 = p1[[2]];,
    If[EvaluateRestriction[p2[[1]], p2[[2]]] ≤ q0 && l[[2]] < ansL,
      ansL = l[[2]];
      ans1 = p2[[1]];
      ans2 = p2[[2]];,
      If[EvaluateRestriction[p3[[1]], p3[[2]]] ≤ q0 && l[[3]] < ansL,
        ansL = l[[3]];
        ans1 = p3[[1]];
        ans2 = p3[[2]];,
        If[EvaluateRestriction[p4[[1]], p4[[2]]] ≤ q0 && l[[4]] < ansL,
          ansL = l[[4]];
          ans1 = p4[[1]];
          ans2 = p4[[2]];
         ];
       ];
     ];
   ];
  ];
  Print[Row[{"Iterations", k}, "="]];
  Print[Row[{"Init 1st block", Row[{1, init1}, "+"]}, "="]];
  Print[Row[{"init 2nd block", Row[{1, init2}, "+"]}, "="]];
  If[ans1 == -1 || ans2 == -1, Print["Failed"],
   Print[Row[{"Add to 1st block", x1 - init1}, "="]];
   Print[Row[{"Add to 2nd block", x2 - init2}, "="]];
   Print[Row[{"Solution for 1st block", Row[{x1, ans1}, "→"]}, "="]];
   Print[Row[{"Solution for 2nd block", Row[{x2, ans2}, "→"]}, "="]];
   Print[Row[{"Probability of failure", EvaluateRestriction[ans1, ans2]}, "="]];
   Print[Row[{"Power consumption", EvaluateTargetF[ans1, ans2]}, "="]];];
 ];
```

Plots

Plotting functions

```
FindNextPoint[cur1_, cur2_, step_] := Module[{grad1, grad2, new1, new2},
  If[EvaluateRestriction[cur1, cur2] ≤ q0,
   grad1 = EvaluateGradTarget1[cur1];
   grad2 = EvaluateGradTarget2[cur2];,
   grad1 = EvaluateGradQ1[cur1];
   grad2 = EvaluateGradQ2[cur2];
  ];
  new1 = cur1 - grad1 * step;
  new2 = cur2 - grad2 * step;
  Return[{new1, new2}];
 ];

FindMinSplit[init1_, init2_, initStep_, ϵ_] := Module[{k = 1, cur1 = init1, cur2 = init2, curF, nextPoint, array, step, prevF, func},
  step = initStep;
  curF = EvaluateTargetF[init1, init2];
  nextPoint = FindNextPoint[init1, init2, step];
  array = {{init1, init2}};
  While[Abs[cur1 - nextPoint[[1]]] ≥ ϵ || Abs[cur2 - nextPoint[[2]]] ≥ ϵ,
   prevF = EvaluateTargetF[cur1, cur2];
   cur1 = nextPoint[[1]];
   cur2 = nextPoint[[2]];
   curF = EvaluateTargetF[cur1, cur2];
   AppendTo[array, {cur1, cur2}];
   nextPoint = FindNextPoint[cur1, cur2, step];
   k++;
   If[curF > prevF, step = initStep / Sqrt[k]];
  ];
  PrintResult[k, init1, init2, cur1, cur2];
  func = Show[ListPlot[array, PlotRange → {{0, 3}, {0, 3}}, PlotStyle → Orange, PlotMarkers → "●"],
    ListPlot[{{array[[1]][[1]], array[[1]][[2]]}}, PlotStyle → Green, PlotMarkers → "●"]];
  Show[PlotAllowedArea[3, 3], func]
 ];
```

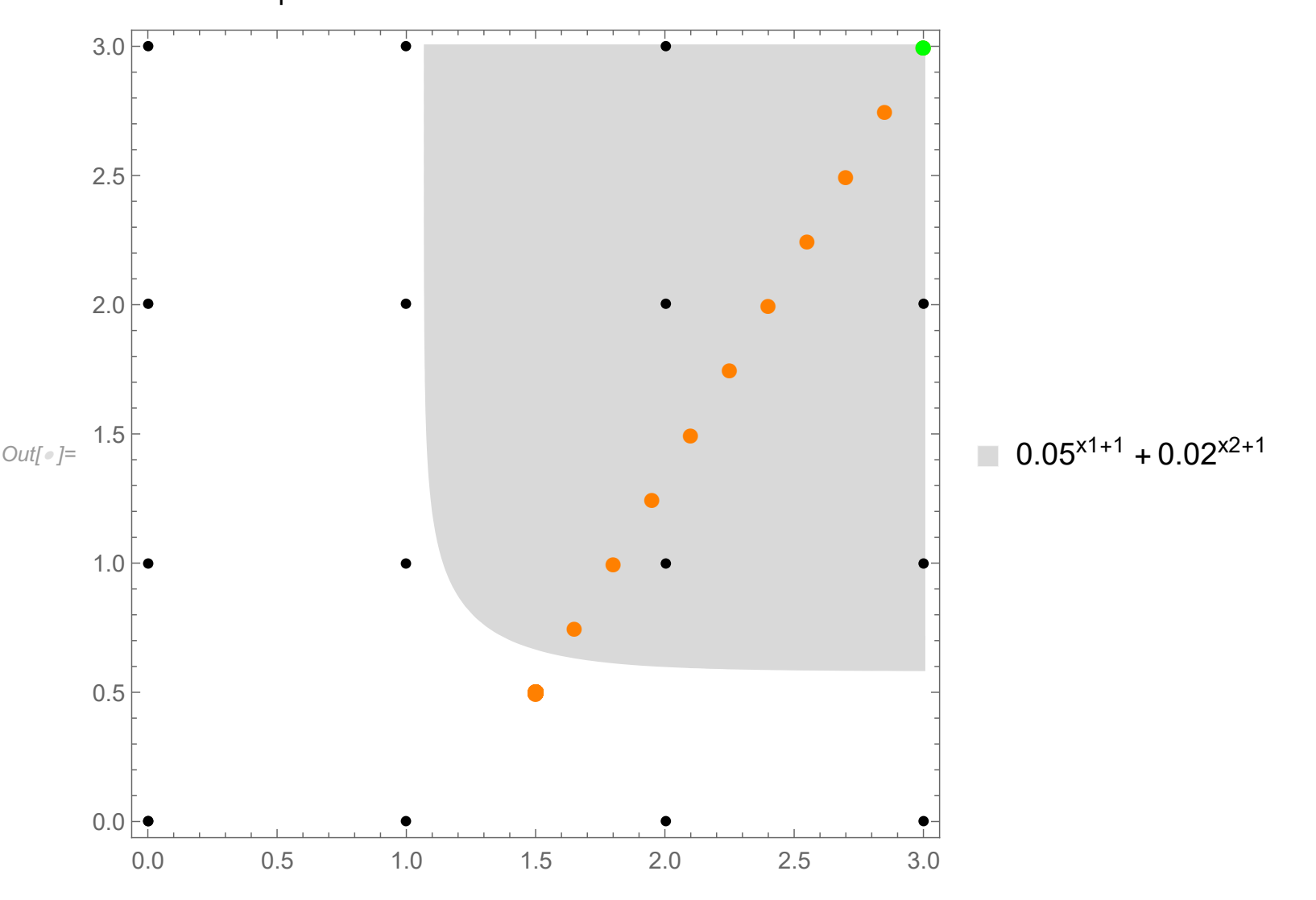
Plots

```
λ = 0.05;
ϵ = 0.00005;
```

Старт из области, где выполняется ограничение

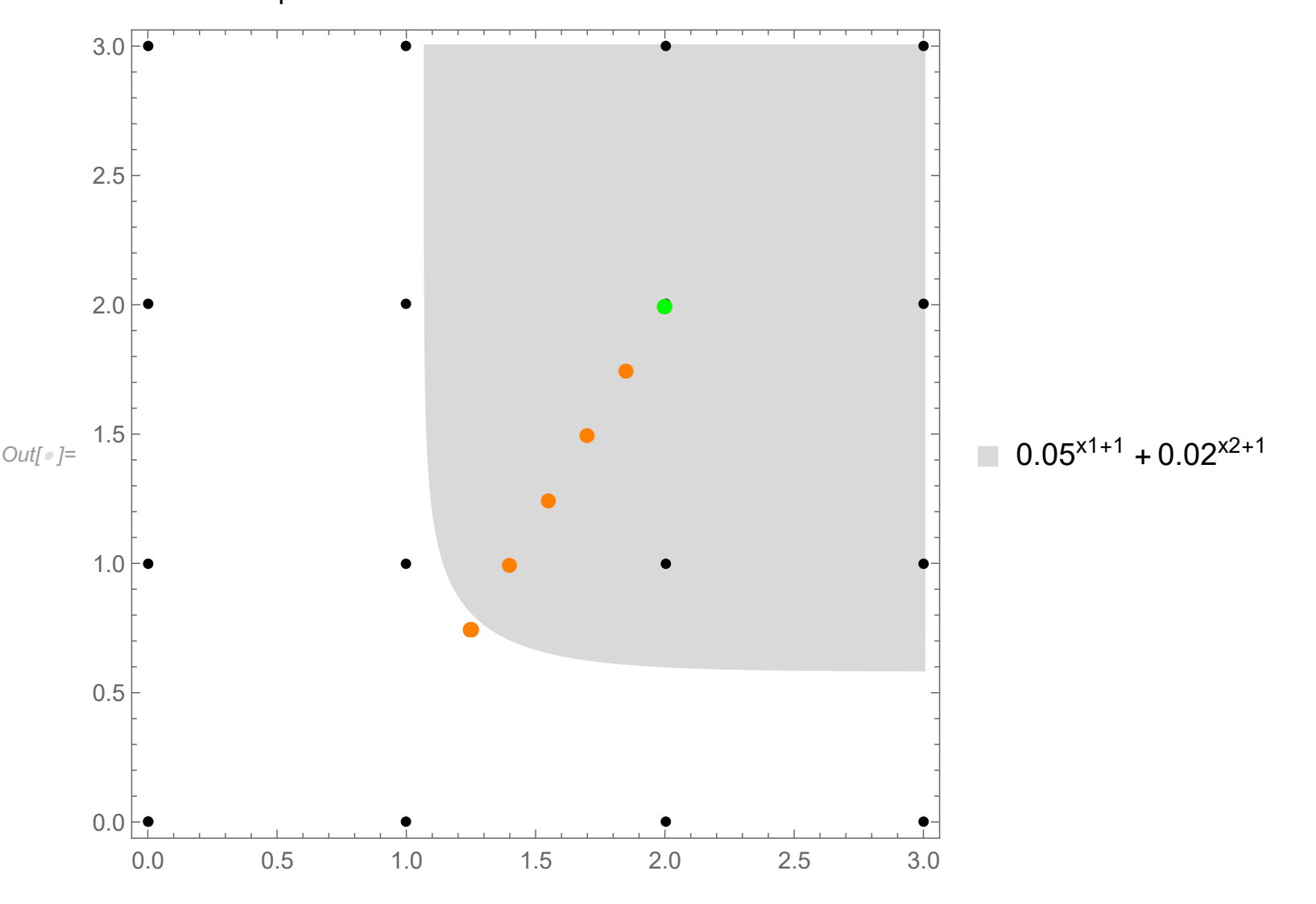
```
FindMinSplit[3, 3, λ, ϵ]
```

```
Iterations = 116
Init 1st block = 1 + 3
init 2nd block = 1 + 3
Add to 1st block = -1.49861
Add to 2nd block = -2.49097
Solution for 1st block = 1.50139 → 2
Solution for 2nd block = 0.509035 → 1
Probability of failure = 0.000525
Power consumption = 19
```



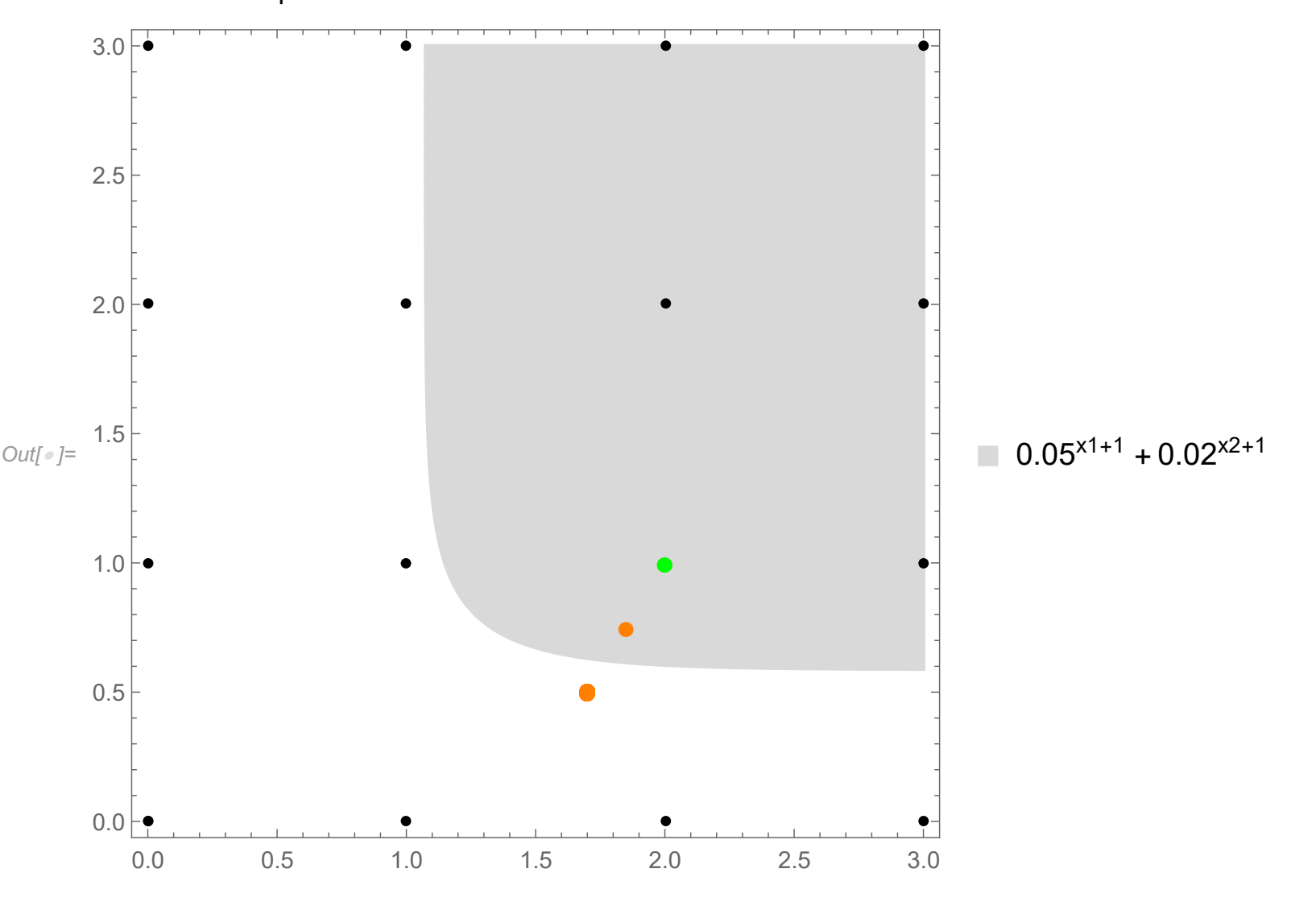

```
FindMinSplit[2, 2, λ, ϵ]
```

```
Iterations = 19
Init 1st block = 1 + 2
init 2nd block = 1 + 2
Add to 1st block = -0.749069
Add to 2nd block = -1.24891
Solution for 1st block = 1.25093 → 2
Solution for 2nd block = 0.751094 → 1
Probability of failure = 0.000525
Power consumption = 19
```




```
FindMinSplit[2, 1, λ, ϵ]
```

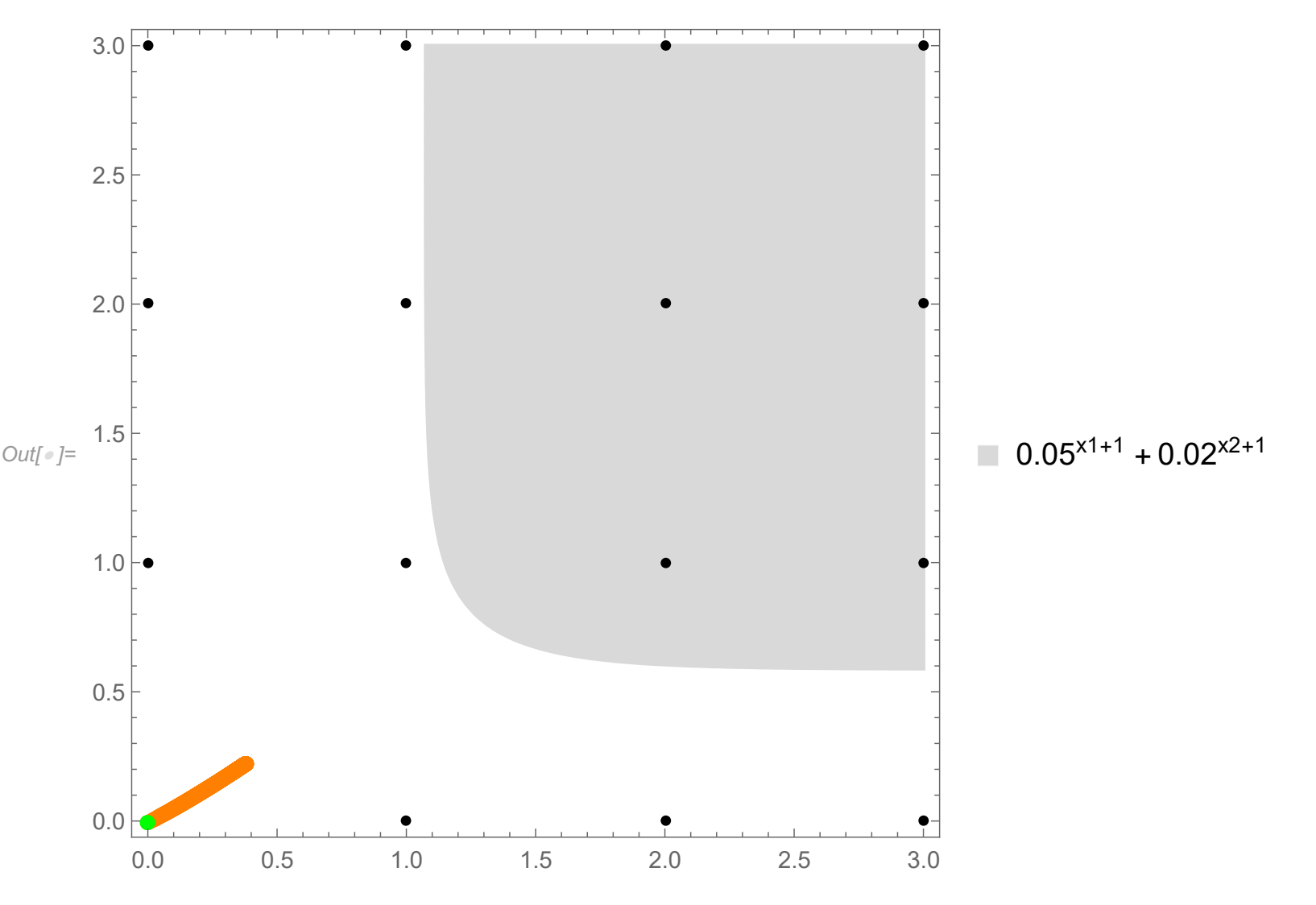
```
Iterations = 114
Init 1st block = 1 + 2
init 2nd block = 1 + 1
Add to 1st block = -0.299106
Add to 2nd block = -0.489447
Solution for 1st block = 1.70089 → 2
Solution for 2nd block = 0.510553 → 1
Probability of failure = 0.000525
Power consumption = 19
```



Старт из области где не выполняется ограничение

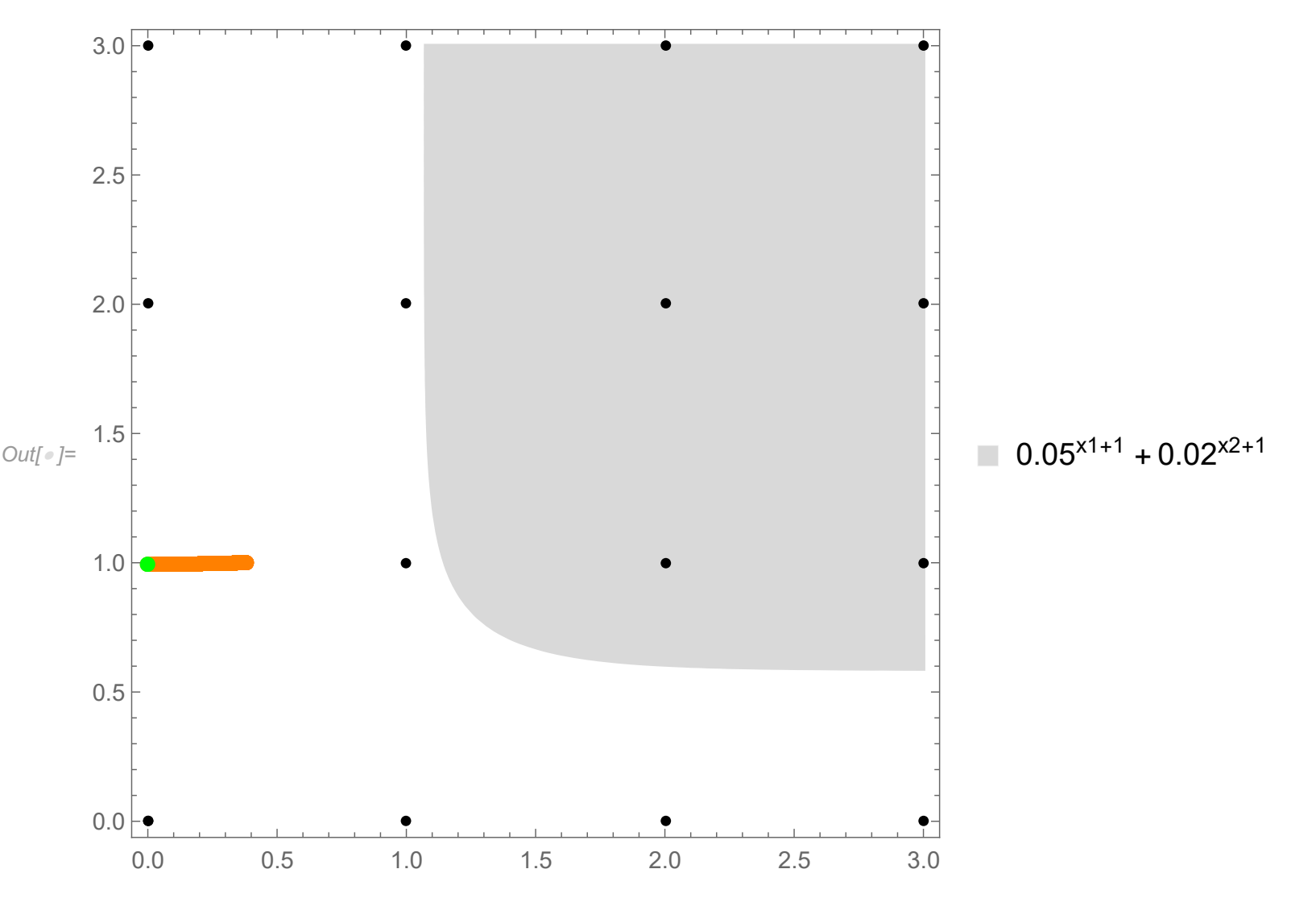
```
FindMinSplit[0, 0, λ, ϵ]
```

```
Iterations = 2284
Init 1st block = 1 + 0
init 2nd block = 1 + 0
Failed
```



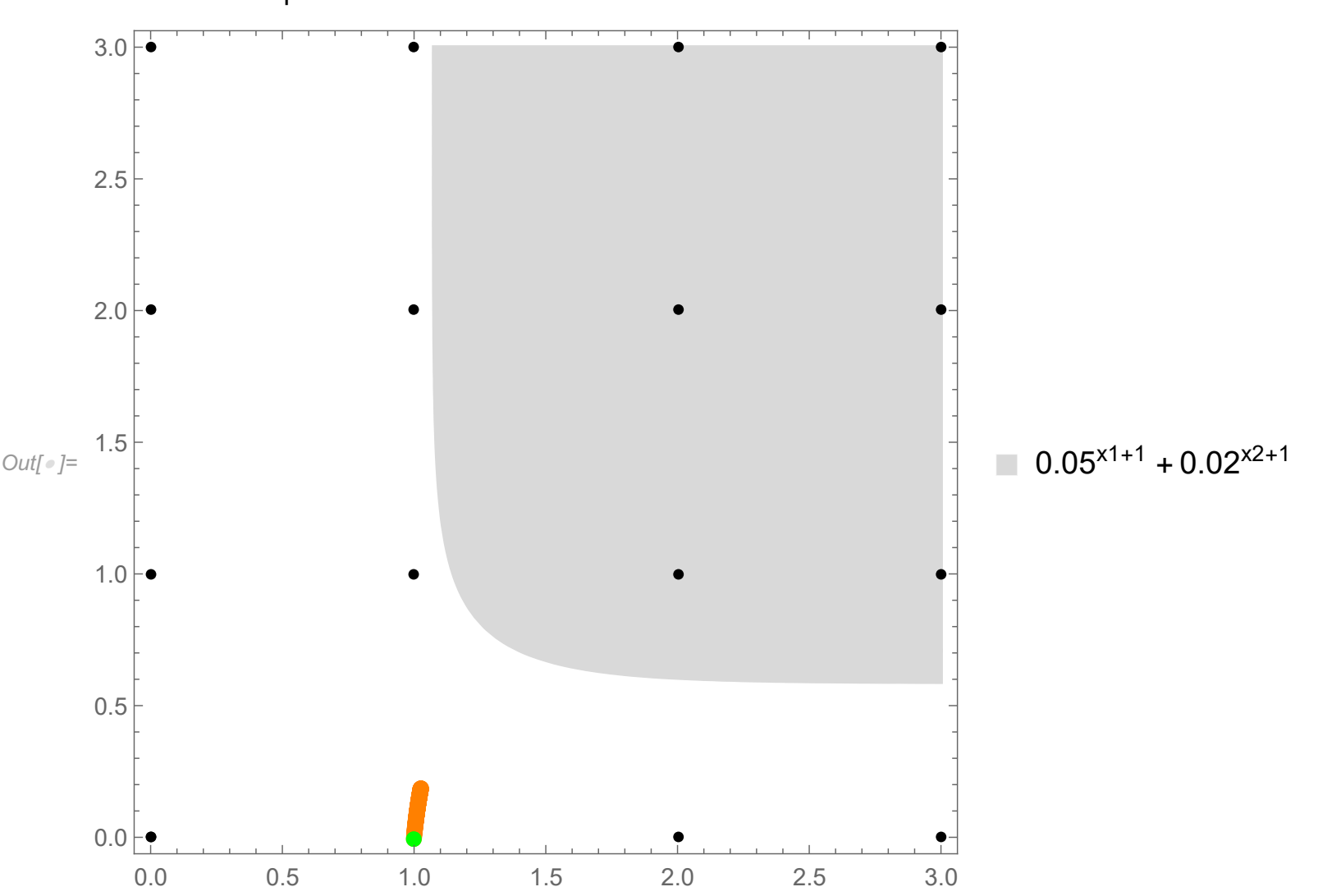

```
FindMinSplit[0, 1, λ, ϵ]
```

```
Iterations = 2284
Init 1st block = 1 + 0
init 2nd block = 1 + 1
Failed
```




```
FindMinSplit[1, 0, λ, ϵ]
```

```
Iterations = 1361
Init 1st block = 1 + 1
init 2nd block = 1 + 0
Add to 1st block = 0.0263738
Add to 2nd block = 0.192365
Solution for 1st block = 1.02637 → 2
Solution for 2nd block = 0.192365 → 1
Probability of failure = 0.000525
Power consumption = 19
```



ВЫВОД:

В лабораторной работе мы познакомились с градиентными методами решения задач нелинейного программирования. В частности, мы находили оптимальное значение функции при помощи градиентных методов. В нашей задаче значения функции дискретны, поэтому задачу решали в предположении, что наша функция гладкая(функция с непрерывными частными производными первого порядка), чтобы мы могли использовать градиентные методы.